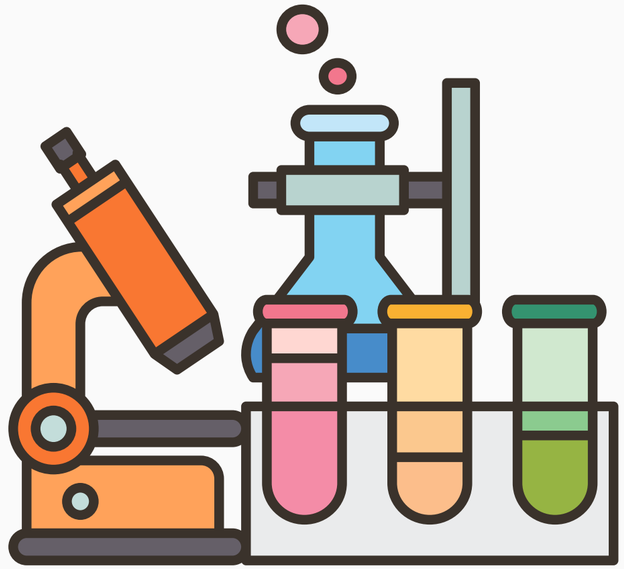
MEDICAMENTOS

VIGENTES

JUAN JOSÉ RINCÓN MÉNDEZ - 2202018
DYLAN FRANCISCO JIMENEZ SANDOVAL - 2202049
SONIA MARCELA GRANADOS MORENO - 2204250

DE QUE TRATA EL DATASET

Trata de medicamentos vigentes y lo que buscamos saber con que laboratorio un medicamento tiene mayor tiempo de duración para tener en cuenta a la hora de comprar.



```
Mounted at /content/drive
```

```
[ ]  1 import pandas as pd
     2 import numpy as np
     3 from datetime import datetime
     4 import datetime as dt
     5 import seaborn as sns
     6 import matplotlib.pyplot as plt
     7 from sklearn.tree import DecisionTreeClassifier
```

Cargar el dataset

```
[ ]  1 datos = pd.read_csv('/content/drive/MyDrive/medicamentos/medicamentos.csv')
     2
     3 datos = datos.replace(',', '.', regex=True)
     4
     5 datos.to_csv("dataset_modificado.csv", index=False)
     6
     7
```

```
[ ]  1 datos['expediente'] = datos['expediente'].astype(int)
     2 datos['producto'] = datos['producto'].astype(str)
```

Importamos las librerías que vamos a utilizar durante el desarrollo del proyecto y subimos el dataset.
Convertimos expediente y producto en datos int.

```
[ ]  1 datos['fechavencimiento'] = pd.to_datetime(datos['fechavencimiento'])
     2 datos['fechaexpedicion'] = pd.to_datetime(datos['fechaexpedicion'])
     3 datos['fechaactivo'] = pd.to_datetime(datos['fechaactivo'])
     4 datos['fechainactivo'] = pd.to_datetime(datos['fechainactivo'])
     5
     6 default_date = pd.Timestamp('00:00:00')
     7 datos['fechavencimiento'] = datos['fechavencimiento'].fillna(default_date)
     8 datos['fechaexpedicion'] = datos['fechaexpedicion'].fillna(default_date)
     9
```

```
[ ]  1 del datos['cantidadcum']
     2 del datos['expedientecum' ]
     3 del datos['concentracion' ]
     4 del datos['descripcioncomercial']
     5 del datos['unidad' ]
     6 del datos['atc' ]
     7 del datos['descripcionatc' ]
     8 del datos['viaadministracion']
     9 del datos['unidadmedida' ]
    10 del datos['unidadreferencia' ]
    11 del datos['formafarmaceutica']
    12 del datos['muestramedica' ]
    13 del datos['cantidad' ]
    14 del datos['IUM' ]
    15 del datos['consecutivocum']
```

Estas líneas eliminan las columnas con los nombres especificados del DataFrame.

Se busca eliminar algunas columnas del Dataset propuesto, ya que estas generan interferencia al momento de realizar predicciones o de entrenar a la maquina.

```
[7]  1 # Eliminar filas duplicadas en el DataFrame
     2 datos = datos.drop_duplicates()
```

```
1 datos.head(50)
```

51	7028	BIODERM® CREMA	LABORATORIOS SIEGFRIED S.A.S.	INVIMA 2023M-005385-R3	19
52	7028	BIODERM® CREMA	LABORATORIOS SIEGFRIED S.A.S.	INVIMA 2023M-005385-R3	19
59	7028	BIODERM® CREMA	LABORATORIOS SIEGFRIED S.A.S.	INVIMA 2023M-005385-R3	19
60	7028	BIODERM® CREMA	LABORATORIOS SIEGFRIED S.A.S.	INVIMA 2023M-005385-R3	19
61	7028	BIODERM® CREMA	LABORATORIOS SIEGFRIED	INVIMA 2023M-	19

Eliminamos las filas duplicadas y las mostramos.

Después de ejecutar estas líneas de código, las columnas 'fechaexpedicion' y 'fechavencimiento' del DataFrame contendrán valores numéricos que representan el timestamp de las fechas originales. Con esto logrando obtener el promedio que es la columna con la cual se busca graficar

```
     2 datos['fechavencimiento'] = datos['fechavencimiento'].apply(lambda x: x.timest
```

Promedio

```
[10]  1 PromedioFechas = datos[['fechaexpedicion', 'fechavencimiento']].mean(axis=1)
      2 datos['Promedio'] = PromedioFechas
      3
```

```
1 datos.head(50)
```

16	5581	VAXOM®CAPSULAS ADULTOS	OM PHARMA SA	INVIMA 2023MB-007838-R4	915
17	5581	BRONCHO-VAXOM®CAPSULAS ADULTOS	OM PHARMA SA	INVIMA 2023MB-007838-R4	915
20	5581	BRONCHO-VAXOM®CAPSULAS ADULTOS	OM PHARMA SA	INVIMA 2023MB-007838-R4	915
21	5581	BRONCHO-VAXOM®CAPSULAS ADULTOS	OM PHARMA SA	INVIMA 2023MB-007838-R4	915

```
[12]  1 datos.shape

(38698, 15)
```

```
[13]  1 datos.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 38698 entries, 0 to 114705
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   expediente         38698 non-null  int64
 1   producto           38698 non-null  object
 2   titular            38698 non-null  object
 3   registrosanitario  38698 non-null  object
 4   fechaexpedicion    38698 non-null  float64
 5   fechavencimiento   38698 non-null  float64
 6   estadoregistro     38698 non-null  object
 7   estadocum          38698 non-null  object
 8   fechaactivo        38698 non-null  datetime64[ns]
 9   fechainactivo      8033 non-null   datetime64[ns]
 10  principioactivo    38698 non-null  object
 11  nombrerol          38698 non-null  object
 12  tiporol            38698 non-null  object
 13  modalidad          38698 non-null  object
 14  Promedio           38698 non-null  float64
dtypes: datetime64[ns](2), float64(3), int64(1), object(9)
memory usage: 4.7+ MB
```

Obtenemos las dimensiones del dataframe, una descripción concisa incluyendo el tipo de datos de cada columna, la cantidad de valores no nulos y la cantidad total de memoria utilizada.

Obtenemos los tipos de datos

```
[14]  1 datos.dtypes

expediente                int64
producto                 object
titular                  object
registrosanitario        object
fechaexpedicion         float64
fechavencimiento        float64
estadoregistro           object
estadocum                object
fechaactivo      datetime64[ns]
fechainactivo    datetime64[ns]
principioactivo          object
nombrerol                object
tiporol                  object
modalidad                object
Promedio                float64
dtype: object
```

```
[15]  1 datos.describe()
```

	expediente	fechaexpedicion	fechavencimiento	Promedio
count	3.869800e+04	3.869800e+04	3.869800e+04	3.869800e+04
mean	1.708055e+07	1.263123e+09	1.785009e+09	1.524066e+09
std	7.078071e+06	2.499299e+08	4.801744e+07	1.265010e+08
min	3.521000e+03	6.005664e+08	1.673309e+09	1.170461e+09
25%	1.993513e+07	1.059091e+09	1.746230e+09	1.422576e+09
50%	2.001021e+07	1.269994e+09	1.784160e+09	1.532390e+09
75%	2.009082e+07	1.460592e+09	1.829520e+09	1.625778e+09

Mostramos los datos

```
[1]  1 datos.expediente.describe()
```

```
count    3.869800e+04
mean     1.708055e+07
std      7.078071e+06
min      3.521000e+03
25%      1.993513e+07
50%      2.001021e+07
75%      2.009082e+07
max      2.024415e+07
Name: expediente, dtype: float64
```

```
[17]  1 datos.columns
```

```
Index(['expediente', 'producto', 'titular', 'registrosanitario',
       'fechaexpedicion', 'fechavencimiento', 'estadoregistro', 'estadocum',
       'fechaactivo', 'fechainactivo', 'principioactivo', 'nombrerol',
       'tiporol', 'modalidad', 'Promedio'],
      dtype='object')
```

```
[18]  1 columnas = datos.columns
      2 columnas
```

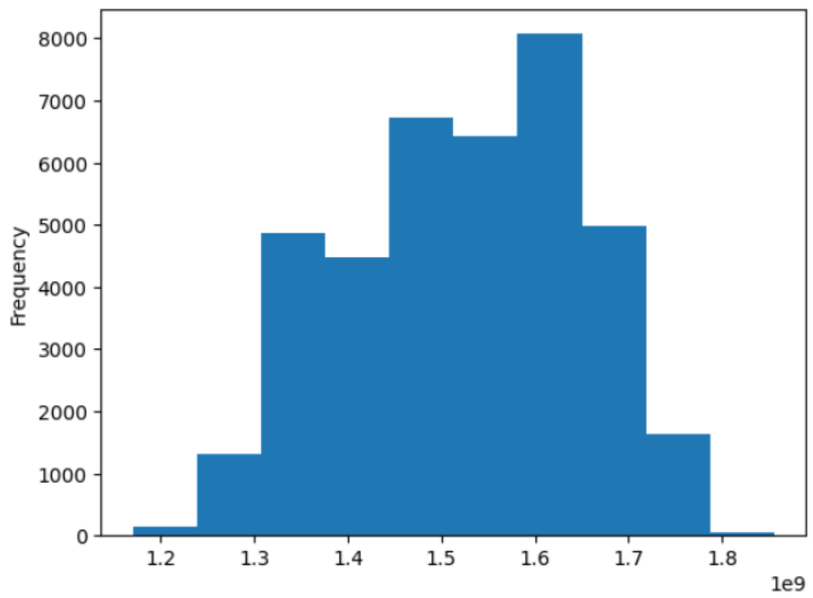
```
Index(['expediente', 'producto', 'titular', 'registrosanitario',
       'fechaexpedicion', 'fechavencimiento', 'estadoregistro', 'estadocum',
       'fechaactivo', 'fechainactivo', 'principioactivo', 'nombrerol',
       'tiporol', 'modalidad', 'Promedio'],
      dtype='object')
```

```
1 datos.tail()
```

	expediente	producto	titular	registrosanitario	fechaexpedicion	fe
114701	20239915	MILACID® SACHET POR 10ML SABOR MENTA	BIOFLUIDOS & FARMA S.A.S.	INVIMA 2023M-0020967	1.693872e+09	
114702	20239915	MILACID® SACHET POR 10ML SABOR MENTA	BIOFLUIDOS & FARMA S.A.S.	INVIMA 2023M-0020967	1.693872e+09	
114703	20239915	MILACID® SACHET POR 10ML SABOR MENTA	BIOFLUIDOS & FARMA S.A.S.	INVIMA 2023M-0020967	1.693872e+09	
114704	20243782	KEPINIA 25 MG	SALUS PHARMA LABS S.A.S.	INVIMA 2023M-0020955	1.680653e+09	
114705	20244153	KEPINIA 100 MG	SALUS PHARMA	INVIMA 2023M-0020954	1.680653e+09	

Obtenemos un resumen estadístico de la columna expediente y las variables.

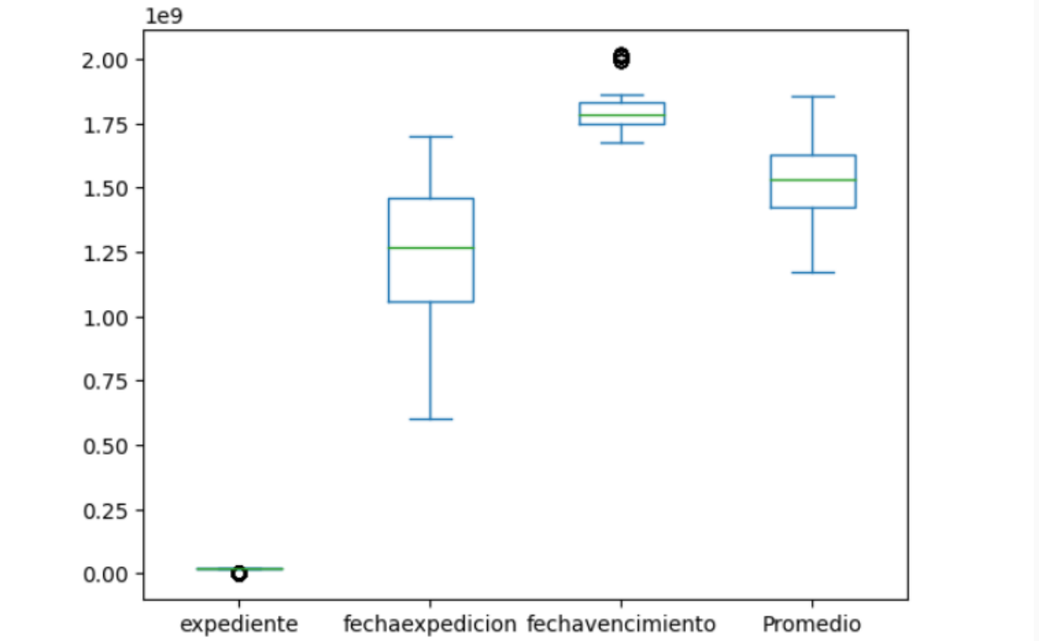
```
1 datos['Promedio'].plot.hist()
```



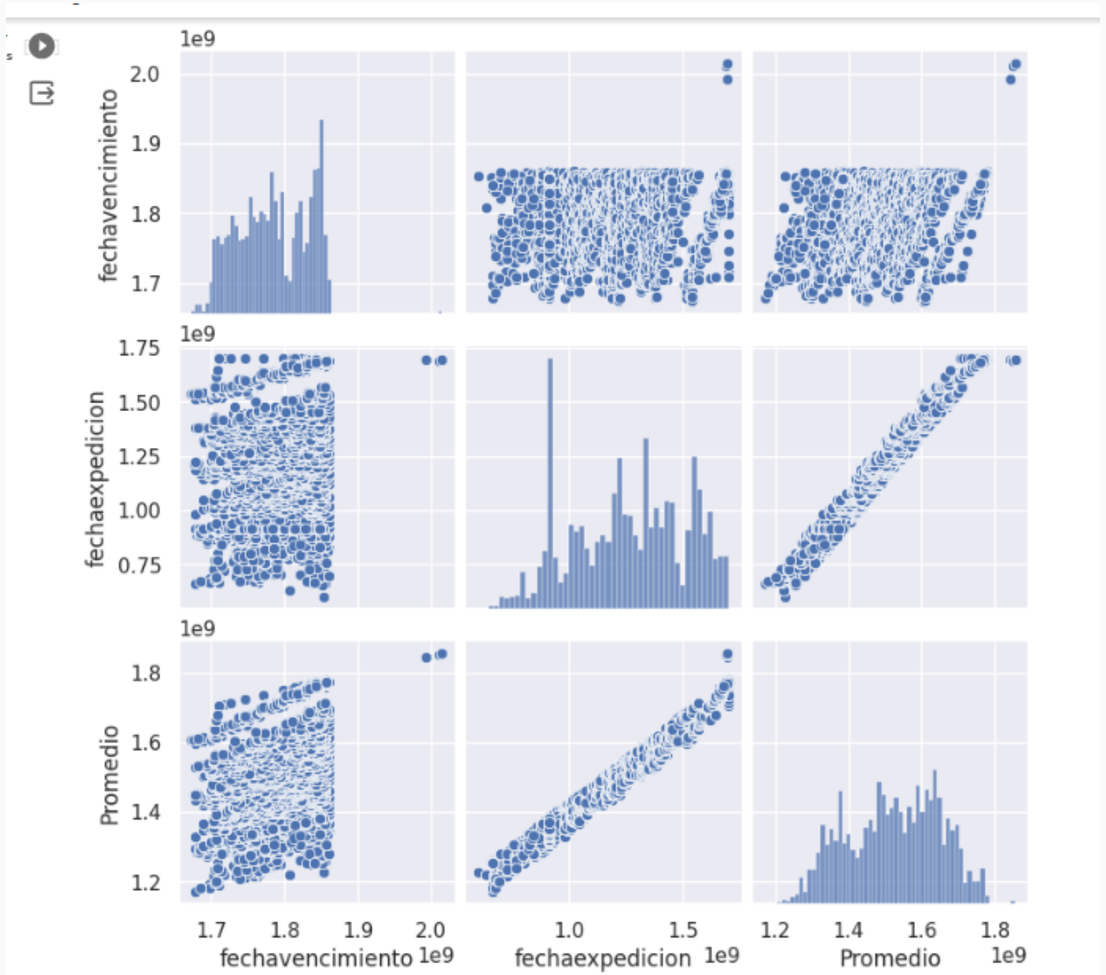
Realizamos un histograma, el cual mostrará la frecuencia de los diferentes valores en la columna 'Promedio'. Cada barra en el histograma representa un rango de valores, y la altura de la barra indica cuántas veces los valores en ese rango aparecen en la columna.

- Realizamos un diagrama de caja que representa el rango intercuartílico (IQR), que es la diferencia entre el tercer cuartil (Q3) y el primer cuartil (Q1). La línea en el medio de la caja es la mediana (Q2).
- Los "bigotes" se extienden hasta los valores que están dentro de 1.5 veces el IQR desde Q1 y Q3. Puntos fuera de estos límites se consideran valores atípicos y se representan como puntos individuales.

```
1 datos.plot.box()
```



```python
X = datos.values[:, 1:]
y = datos.values[:, [1]]

sns.set(rc={'figure.figsize': (15, 15)})

sns.pairplot(datos, diag_kind="hist")

plt.show()

print(X.shape, y.shape)

datos.head(5)

columns = ["fechavencimiento", "fechaexpedicion", "Promedio"]
X = datos[columns].values

sns.set(rc={'figure.figsize': (15, 15)})

sns.pairplot(datos[columns], diag_kind="hist")  # kind="kde"

plt.show()

print(X.shape)
datos[columns].head(5)
```



(38698, 3)

	fechavencimiento	fechaexpedicion	Promedio
0	1.794874e+09	945648000.0	1.370261e+09
1	1.794874e+09	945648000.0	1.370261e+09
2	1.794874e+09	945648000.0	1.370261e+09
8	1.794874e+09	945648000.0	1.370261e+09
11	1.794874e+09	945648000.0	1.370261e+09

Visualizamos la distribución y las relaciones entre las variables del DataFrame utilizando pair plots, además de seleccionar columnas específicas para un análisis más detallado.

```
k=0
for i in datos.producto.unique():
    datos.producto.replace(i, k, inplace = True)
    k = k+1
```

```
datos.head(10)
```

	expediente	producto	titular	registrosanitario	fechaexpedicion	fechavenc
0	3521	0	PROCAPS S.A.	INVIMA 2021M-002103-R3	945648000.0	1.794
1	3521	0	PROCAPS S.A.	INVIMA 2021M-002103-R3	945648000.0	1.794
2	3521	0	PROCAPS S.A.	INVIMA 2021M-002103-R3	945648000.0	1.794
8	3521	0	PROCAPS S.A.	INVIMA 2021M-002103-R3	945648000.0	1.794
11	3521	0	PROCAPS S.A.	INVIMA 2021M-002103-R3	945648000.0	1.794
12	3521	0	PROCAPS S.A.	INVIMA 2021M-002103-R3	945648000.0	1.794
13	3521	0	PROCAPS S.A.	INVIMA 2021M-002103-R3	945648000.0	1.794
16	5581	1	OM PHARMA SA	INVIMA 2023MB-007838-R4	915148800.0	1.851

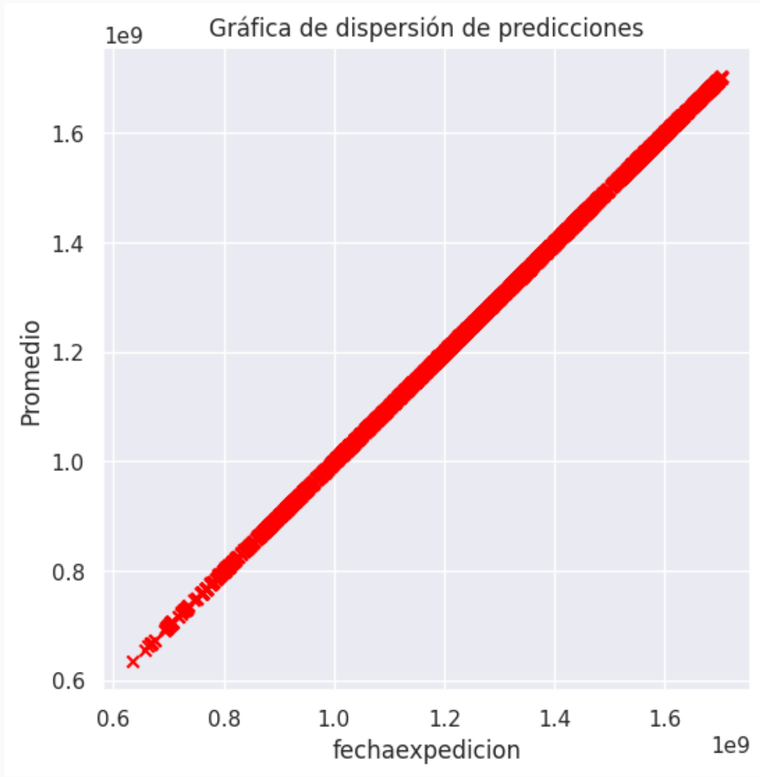
Este código proporciona una visualización del Dataset cono los cambios realizados, donde se puede observar el cambio de fecha a un valor numerico.

Matriz de correlación

```
1 corr = datos.corr()
2 plt.subplots(figsize = (25,15))
3 sns.heatmap(corr, annot = True, fmt = '.2f')
```

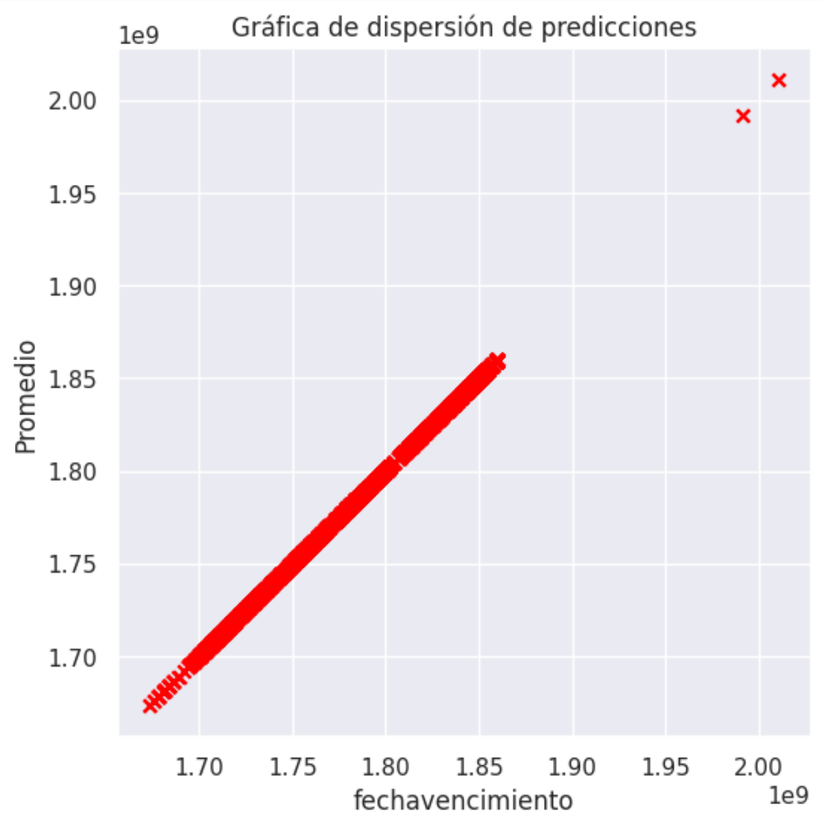
Los valores en el mapa de calor indican la fuerza y la dirección de la relación lineal entre las diferentes variables. El valor cercano a 1 indica una correlación positiva fuerte, mientras que el valor cercano a -1 indica una correlación negativa fuerte. El valor cercano a 0 indica una correlación débil.

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


X = datos[['fechaexpedicion']]
y = datos[['Promedio']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
y_train = np.array(y_train).ravel()


est = GaussianNB()

est.fit(X_train,y_train)
print("%.3f" % accuracy_score(est.predict(X_test), y_test))

fig = plt.figure(figsize=(6, 6))
plt.scatter(X_test['fechaexpedicion'], X_test['fechaexpedicion'], c="red" , cmap='coolwarm', marker='x')
plt.xlabel('fechaexpedicion')
plt.ylabel('Promedio')
plt.title('Gráfica de dispersión de predicciones')
plt.show()
```



Este código utiliza un Regresor Gaussiano para predecir 'Promedio' basado en la característica 'fechaexpedicion'. La precisión del modelo se imprime y se visualizan las predicciones en un gráfico de dispersión

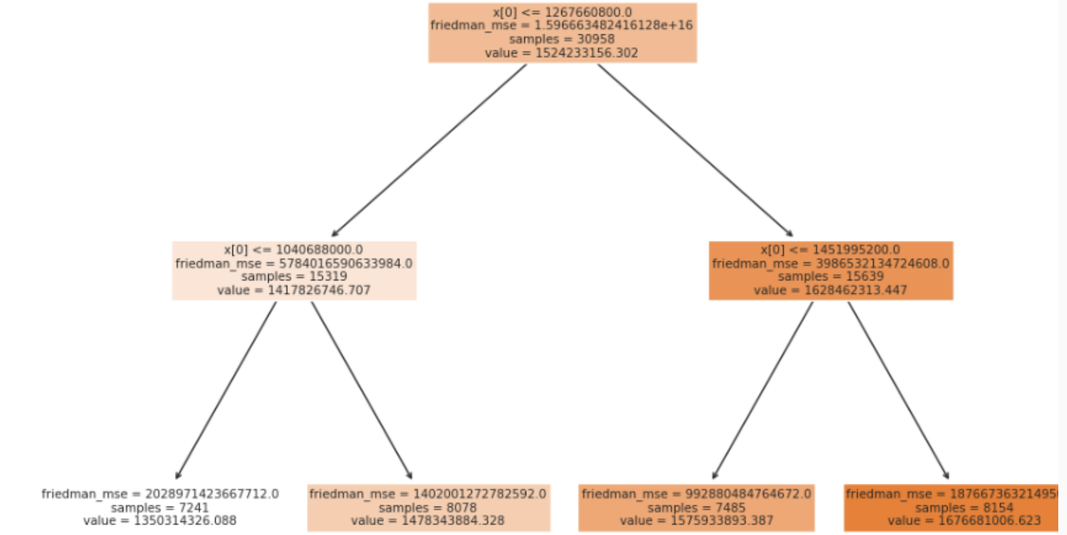
```python
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


X = datos[['fechavencimiento']]
y = datos[['Promedio']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
y_train = np.array(y_train).ravel()


est = GaussianNB()

est.fit(X_train,y_train)
print("%.3f" % accuracy_score(est.predict(X_test), y_test))

fig = plt.figure(figsize=(6, 6))
plt.scatter(X_test['fechavencimiento'], X_test['fechavencimiento'], c='red', cmap='coolwarm', marker='x')
plt.xlabel('fechavencimiento')
plt.ylabel('Promedio')
plt.title('Gráfica de dispersión de predicciones')
plt.show()
```

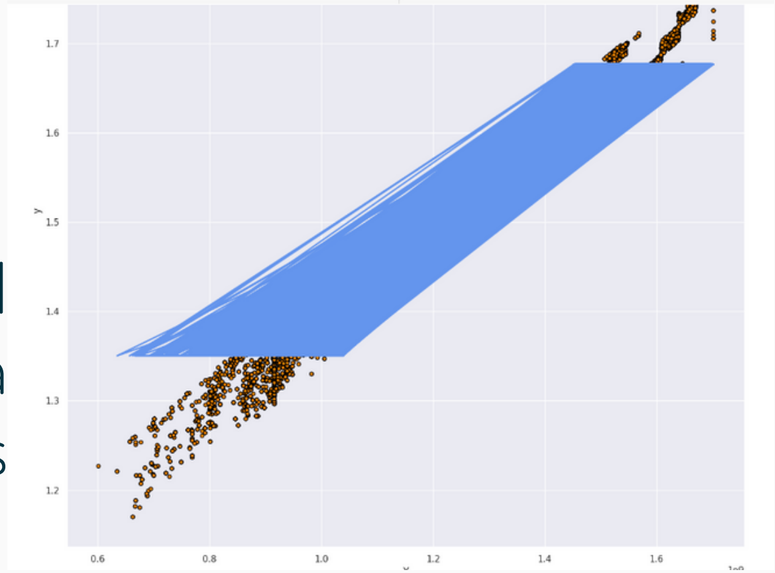


- Cada punto en la gráfica representa una instancia del conjunto de prueba.
- El color de los puntos (marcadores 'x') indica las predicciones del modelo. Los colores más cálidos o más fríos podrían representar diferentes valores predichos.

```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error, mean_absolute_error

X = datos[['fechaexpedicion']]
y = datos[['Promedio']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

max_depth = 2
criterion = 'friedman_mse'
clf = DecisionTreeRegressor(max_depth=max_depth, criterion=criterion)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print("MSE :", mse)

rmse = np.sqrt(mse)
print("RMSE:", rmse)

y_pred = clf.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
print("MAE :", mae)

from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True)
plt.show()

plt.figure()
plt.scatter(X, y, s=20, edgecolor="black", c="darkorange", label="data")
plt.plot(X_test, y_pred, color="cornflowerblue", linewidth=2, label="regression")
plt.xlabel("X")
plt.ylabel("y")
plt.title("Decision Tree Regression")
plt.legend()
```

```
MSE : 1566960653612978.2
RMSE: 39584853.84099553
MAE : 31748084.99130711
```

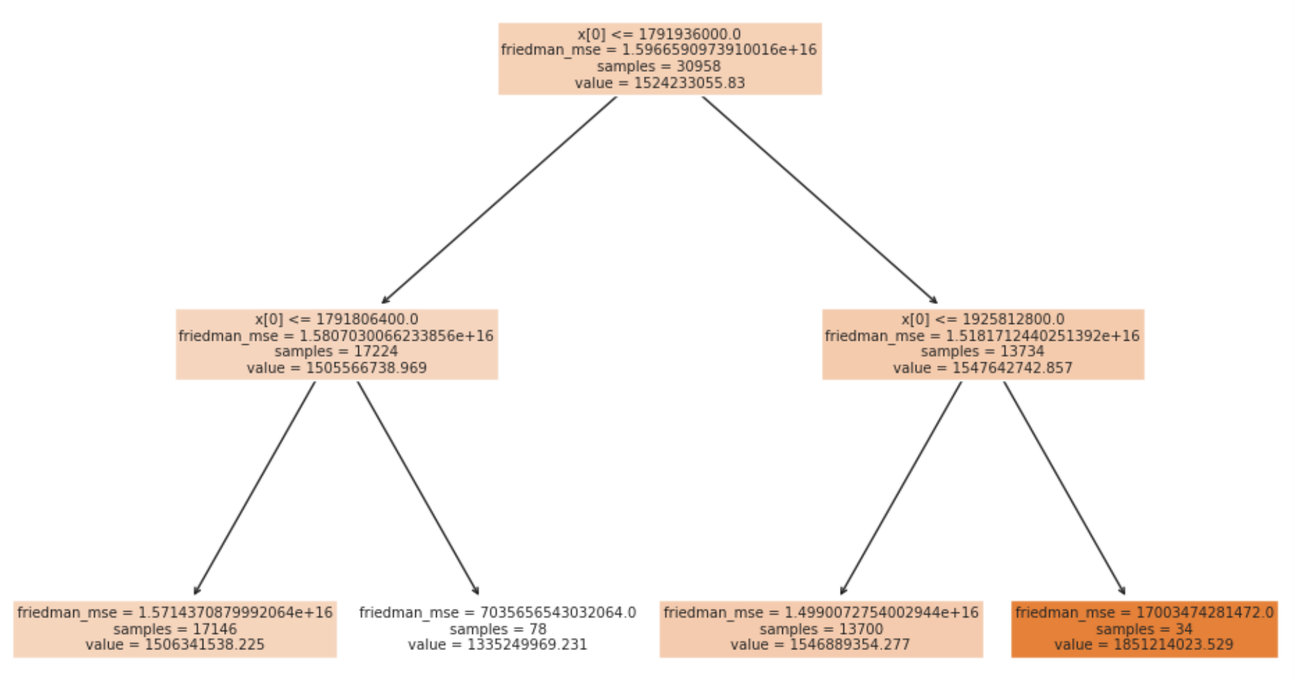




Este código realiza un análisis de regresión utilizando un árbol de decisión, evalúa su rendimiento y proporciona visualizaciones para entender mejor tanto el modelo como sus decisiones.

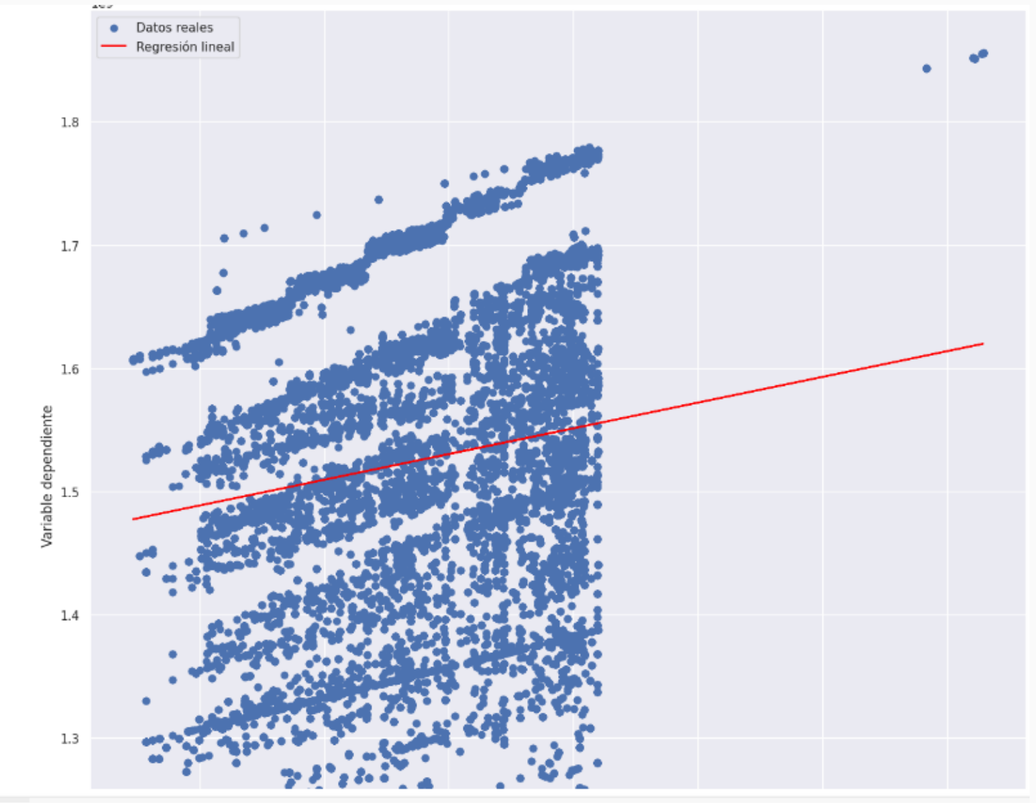
```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error, mean_absolute_error

X = datos[['fechavencimiento']]
y = datos[['Promedio']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

max_depth = 2
criterion = 'friedman_mse'
clf = DecisionTreeRegressor(max_depth=max_depth, criterion=criterion)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print("MSE :", mse)

rmse = np.sqrt(mse)
print("RMSE: ", rmse)

y_pred = clf.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
print("MAE :", mae)

from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True)
plt.show()
```

```
MSE : 1.5508963703328376e+16
RMSE:  124534989.87565051
MAE : 105395147.13827674
```



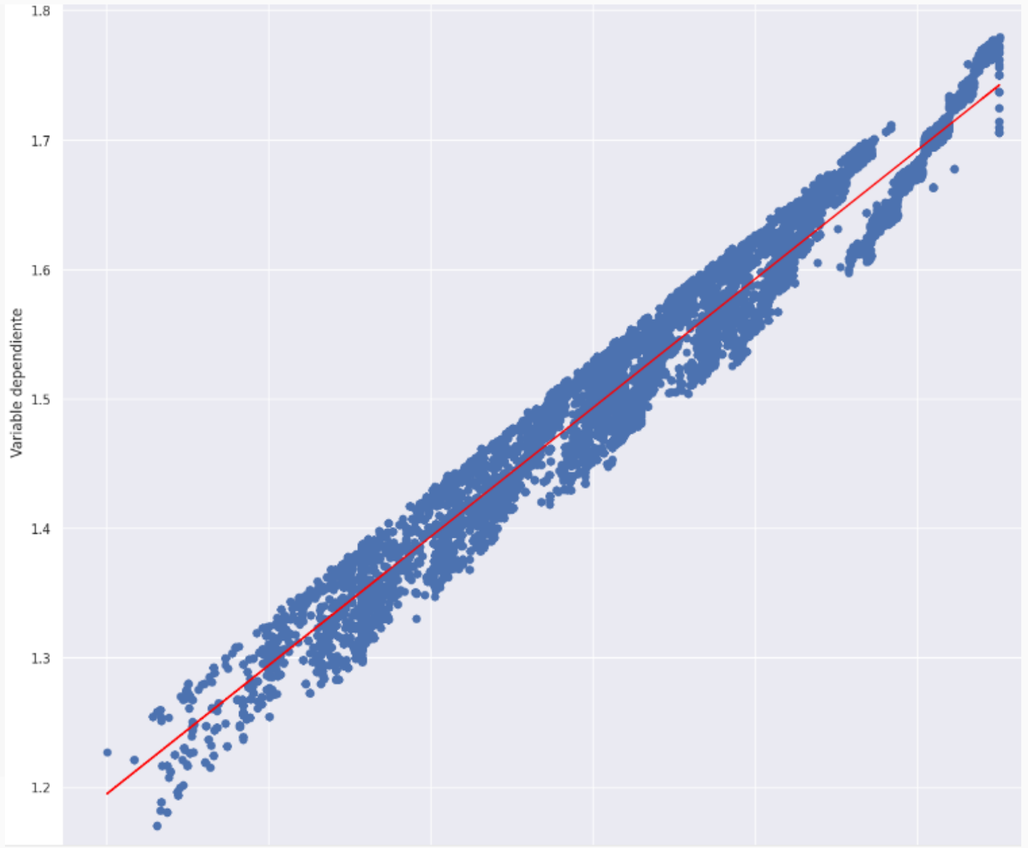
Este codigo realiza un análisis de regresión utilizando un árbol de decisión, evalúa su rendimiento y proporciona visualizaciones para entender mejor tanto el modelo como sus decisiones específicas en relación con la fecha de vencimiento.

```python
import numpy as np
from sklearn.linear_model import LinearRegression

X = datos[['fechavencimiento']]
y = datos[['Promedio']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
slope = model.coef_[0]
intercept = model.intercept_
r_squared = model.score(X, y)

plt.scatter(X, y, label='Datos reales')
plt.plot(X, y_pred, color='red', label='Regresión lineal')
plt.legend()
plt.xlabel('Variable independiente')
plt.ylabel('Variable dependiente')
plt.show()
```



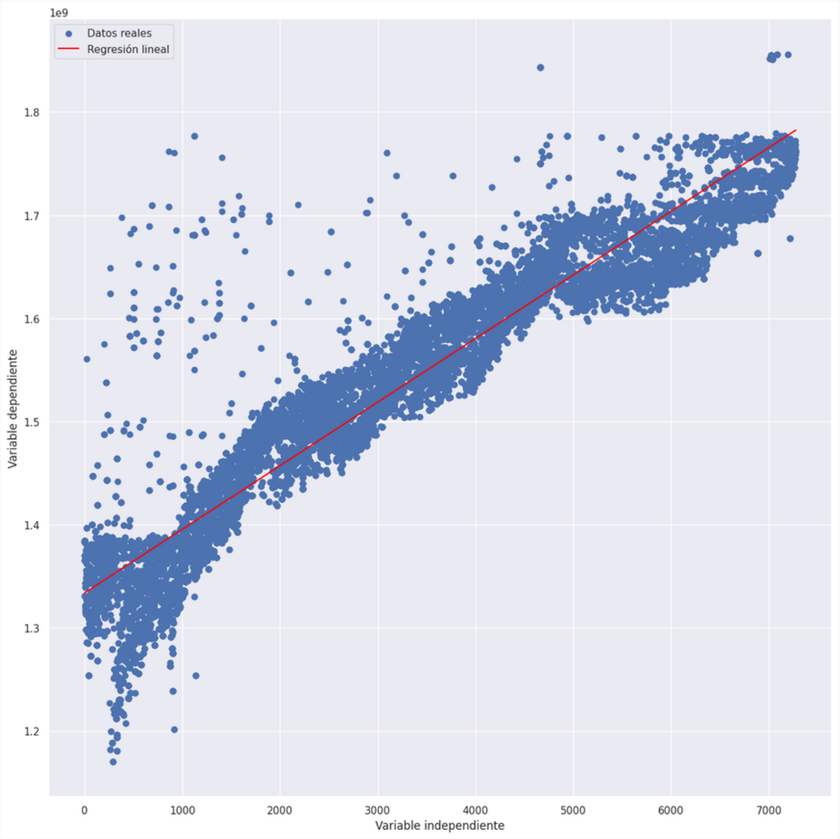
Este código utiliza un modelo de regresión lineal simple para modelar la relación entre la fecha de vencimiento y el promedio. Calcula algunos parámetros clave y visualiza la regresión lineal resultante. Este enfoque es útil para determinar si la relación entre las variables es lineal.

```python
import numpy as np
from sklearn.linear_model import LinearRegression

X = datos[['fechaexpedicion']]
y = datos[['Promedio']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
slope = model.coef_[0]
intercept = model.intercept_
r_squared = model.score(X, y)

plt.scatter(X, y, label='Datos reales')
plt.plot(X, y_pred, color='red', label='Regresión lineal')
plt.legend()
plt.xlabel('Variable independiente')
plt.ylabel('Variable dependiente')
plt.show()
```



Este código utiliza un modelo de regresión lineal simple para modelar la relación entre la fecha de vencimiento y el promedio. Calcula algunos parámetros clave y visualiza la regresión lineal resultante. Este enfoque es útil para determinar si la relación entre las variables es lineal.

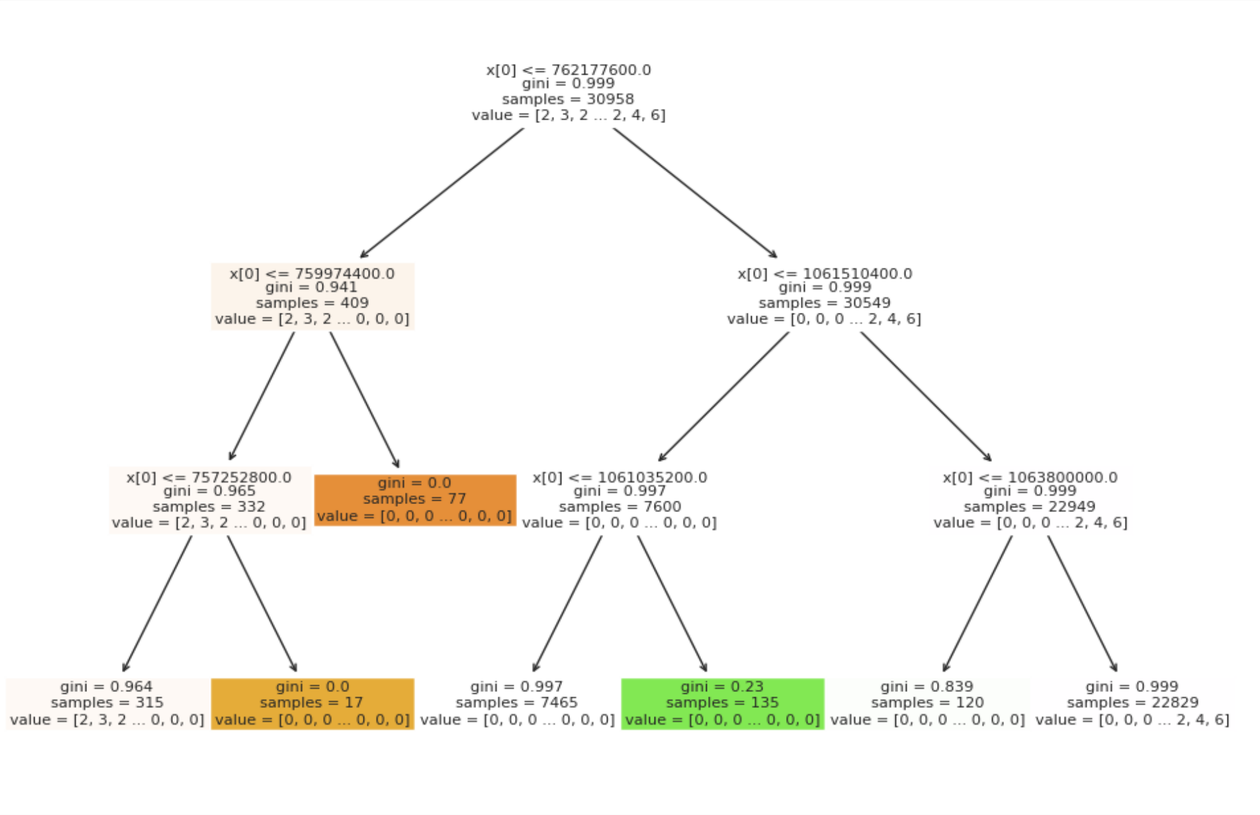
```python
import numpy as np
from sklearn.linear_model import LinearRegression

X = datos[['producto']]
y = datos[['Promedio']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
slope = model.coef_[0]
intercept = model.intercept_
r_squared = model.score(X, y)

plt.scatter(X, y, label='Datos reales')
plt.plot(X, y_pred, color='red', label='Regresión lineal')
plt.legend()
plt.xlabel('Variable independiente')
plt.ylabel('Variable dependiente')
plt.show()
```



Este código utiliza un modelo de regresión lineal simple para modelar la relación entre el producto y el promedio. Calcula algunos parámetros clave y visualiza la regresión lineal resultante. Este enfoque es útil para determinar si la relación entre las variables es lineal.

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

X = datos[['fechaexpedicion']]
y = datos[['Promedio']]
np.random.seed(20)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

max_depth = 3
criterion = 'gini'
clf = DecisionTreeClassifier(max_depth=max_depth, criterion=criterion)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo: {accuracy}")

plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True)
plt.show()
```

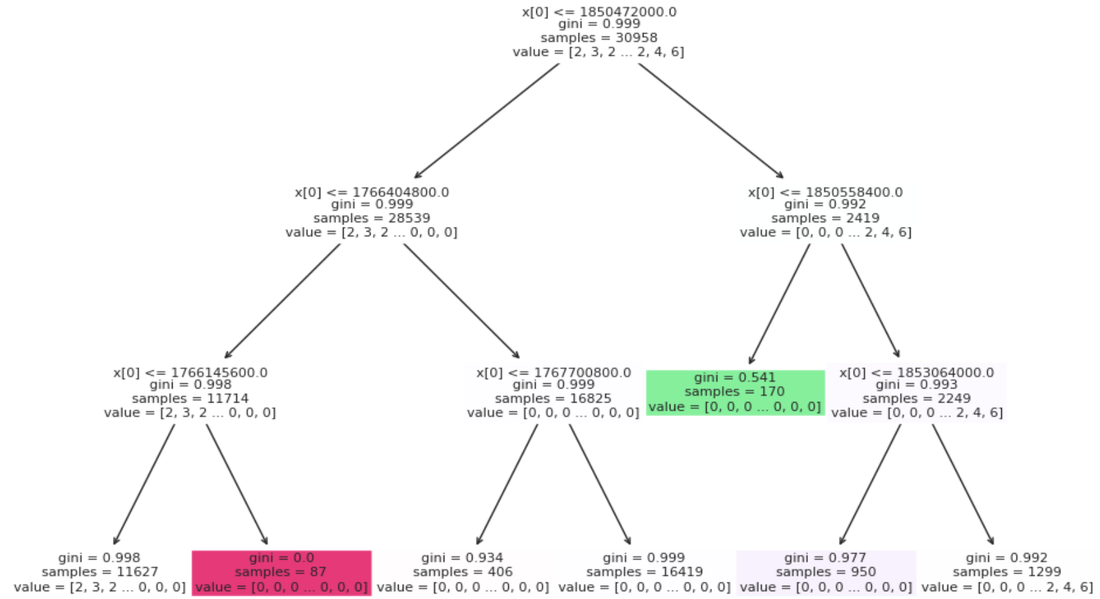
```
Precisión del modelo: 0.017054263565891473
```



Este código utiliza un Decision Tree Classifier para predecir la variable Promedio basándose en la fecha de expedición. Luego, evalúa la precisión del modelo y visualiza el árbol de decisión para entender cómo toma decisiones.

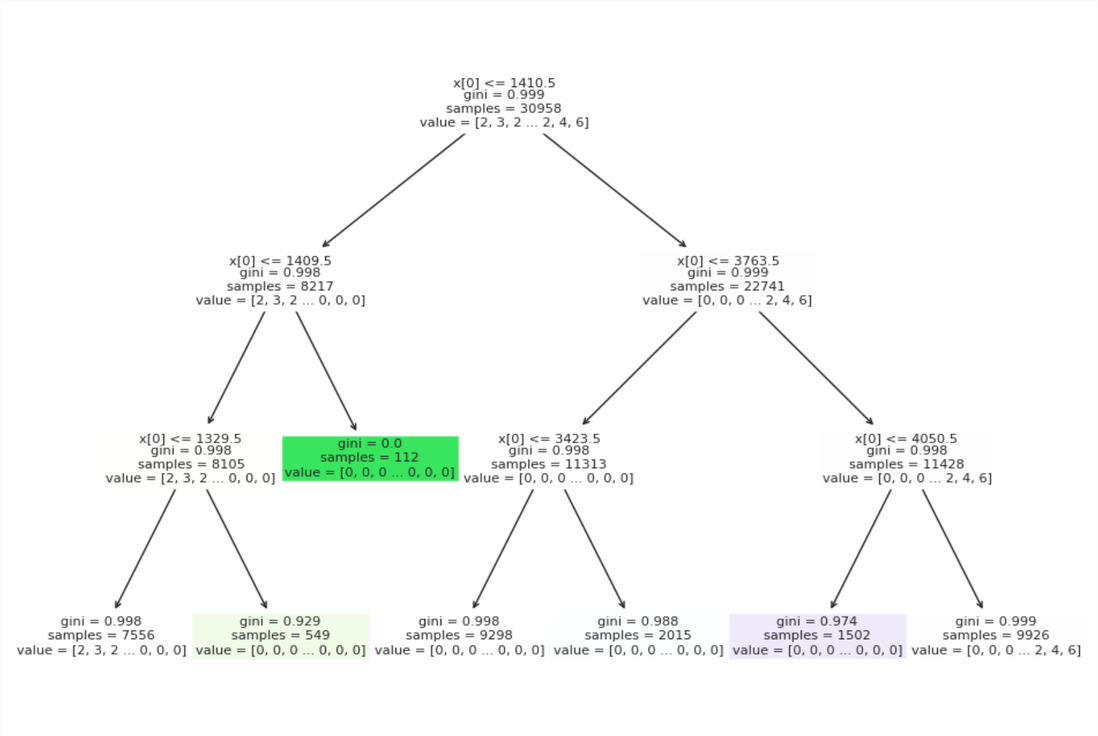
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X = datos[['fechavencimiento']]
y = datos[['Promedio']]
np.random.seed(20)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

max_depth = 3
criterion = 'gini'
clf = DecisionTreeClassifier(max_depth=max_depth, criterion=criterion)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo: {accuracy}")

from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True)
plt.show()

```

```
Precisión del modelo: 0.021576227390180877
```



Este código utiliza un Decision Tree Classifier para predecir la variable Promedio basándose en la fecha de vencimiento (fechavencimiento). Evalúa la precisión del modelo y proporciona una visualización del árbol de decisión para entender cómo toma decisiones.

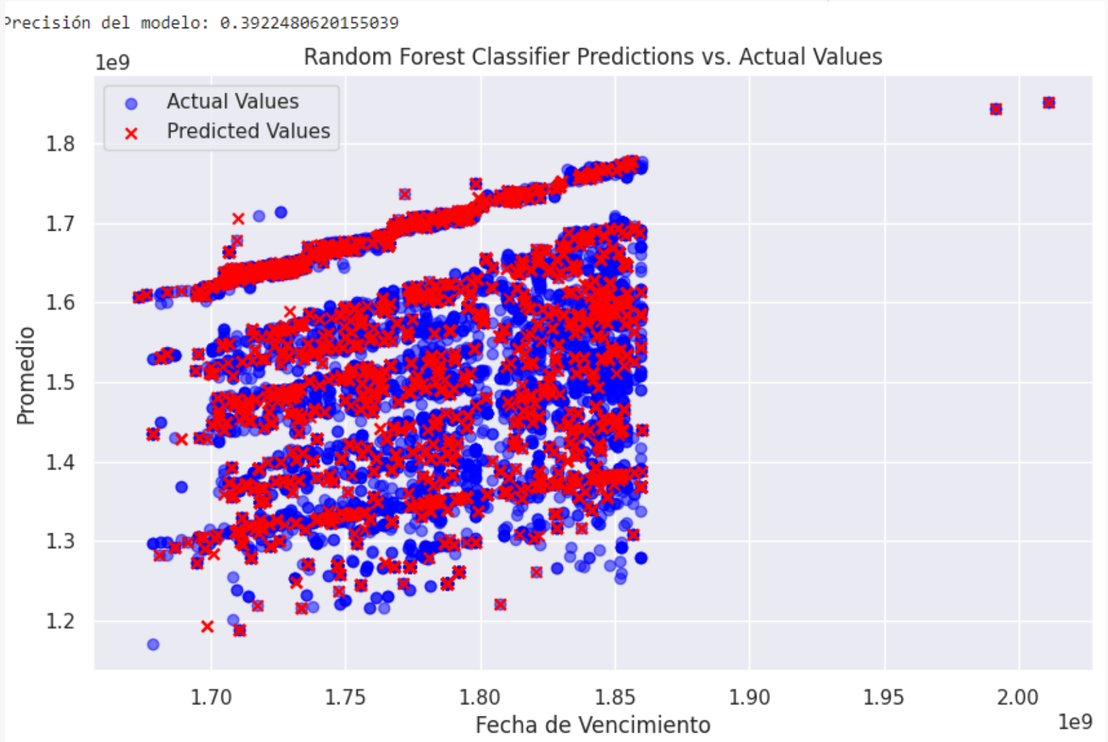
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

X = datos[['producto']]
y = datos[['Promedio']]
np.random.seed(20)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

max_depth = 3
criterion = 'gini'
clf = DecisionTreeClassifier(max_depth=max_depth, criterion=criterion)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo: {accuracy}")

plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True)
plt.show()
```



Este código utiliza un Decision Tree Classifier para predecir la variable Promedio basándose en la columna producto. Evalúa la precisión del modelo y proporciona una visualización del árbol de decisión para entender cómo toma decisiones.

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X = datos[['fechavencimiento']]
y = datos[['Promedio']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

y_train = np.array(y_train).ravel()
n_estimators = 100
max_depth = 400
criterion = 'gini'
clf = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth, criterion=criterion)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo: {accuracy}")

plt.show()

plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual Values', alpha=0.5)
plt.scatter(X_test, y_pred, color='red', marker='x', label='Predicted Values')
plt.xlabel('Fecha de Vencimiento')
plt.ylabel('Promedio')
plt.legend()
plt.title('Random Forest Classifier Predictions vs. Actual Values')
plt.show()
```

```
Precisión del modelo: 0.3922480620155039
```



Este código utiliza un Random Forest Classifier o para predecir la variable Promedio basándose en la fecha de vencimiento (fechavencimiento). Evalúa la precisión del modelo y proporciona una visualización de cómo se comparan las predicciones con los valores reales. El bosque aleatorio es una técnica de ensamble que utiliza múltiples árboles de decisión para mejorar la precisión del modelo.

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X = datos[['fechaexpedicion']]
y = datos[['Promedio']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

y_train = np.array(y_train).ravel()
n_estimators = 100
max_depth = 400
criterion = 'gini'
clf = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth, criterion=criterion)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo: {accuracy}")

plt.show()

plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual Values', alpha=0.5)
plt.scatter(X_test, y_pred, color='red', marker='x', label='Predicted Values')
plt.xlabel('Fecha de Expedicion')
plt.ylabel('Promedio')
plt.legend()
plt.title('Random Forest Classifier Predictions vs. Actual Values')
plt.show()
```

Este código utiliza un Random Forest Classifier para predecir la variable Promedio basándose en la fecha de expedición (fechaexpedicion). Evalúa la precisión del modelo y proporciona una visualización de cómo se comparan las predicciones con los valores reales. El bosque aleatorio es una técnica de ensamble que utiliza múltiples árboles de decisión para mejorar la precisión del modelo.

PCA

```python
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

columna_pca = ['producto', 'Promedio']

data_for_pca = datos[columna_pca]

mean = np.mean(data_for_pca)
std = np.std(data_for_pca)
print("mean:", mean)
print("std:", std)

standardized_data = (data_for_pca - mean) / std
```

```
mean: producto    3.086220e+03
Promedio    1.524067e+09
dtype: float64
std: producto    1.944146e+03
Promedio    1.264999e+08
dtype: float64
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:3430: FutureWarning: In a future version, DataFrame.mean(axis=None) will return a scalar mean over the entire DataFrame. To retain the old behavior, use 'frame.mean(axis=0)' or
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

```python
standardized_data
```

	producto	Promedio
0	-1.587443	-1.215856
1	-1.587443	-1.215856
2	-1.587443	-1.215856
8	-1.587443	-1.215856
11	-1.587443	-1.215856
...	...	...
114701	2.154046	1.931087
114702	2.154046	1.931087
114703	2.154046	1.931087

Se busca obtener la covarianza de la matriz por medio de la estandarizacion, para despues reducir la data

```
covariance_matrix = np.cov(standardized_data, ddof = 1, rowvar = False)
covariance_matrix
```

```
array([[1.00002584, 0.94747949],
       [0.94747949, 1.00002584]])
```

```
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

print("eigenvalues", eigenvalues)

print("eigenvectors", eigenvectors)
```

```
eigenvalues [1.94750533 0.05254635]
eigenvectors [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```

```
# np.argsort can only provide lowest to highest; use [::-1] to reverse the list
order_of_importance = np.argsort(eigenvalues)[::-1]

# utilize the sort order to sort eigenvalues and eigenvectors
sorted_eigenvalues = eigenvalues[order_of_importance]
sorted_eigenvectors = eigenvectors[:,order_of_importance] # sort the columns
```

```
# use sorted_eigenvalues to ensure the explained variances correspond to the eigenvectors
explained_variance = sorted_eigenvalues / np.sum(sorted_eigenvalues)
```

```
k = 2 # select the number of principal components
reduced_data = np.matmul(standardized_data, sorted_eigenvectors[:,:k]) # transform the original data

print(reduced_data)
```

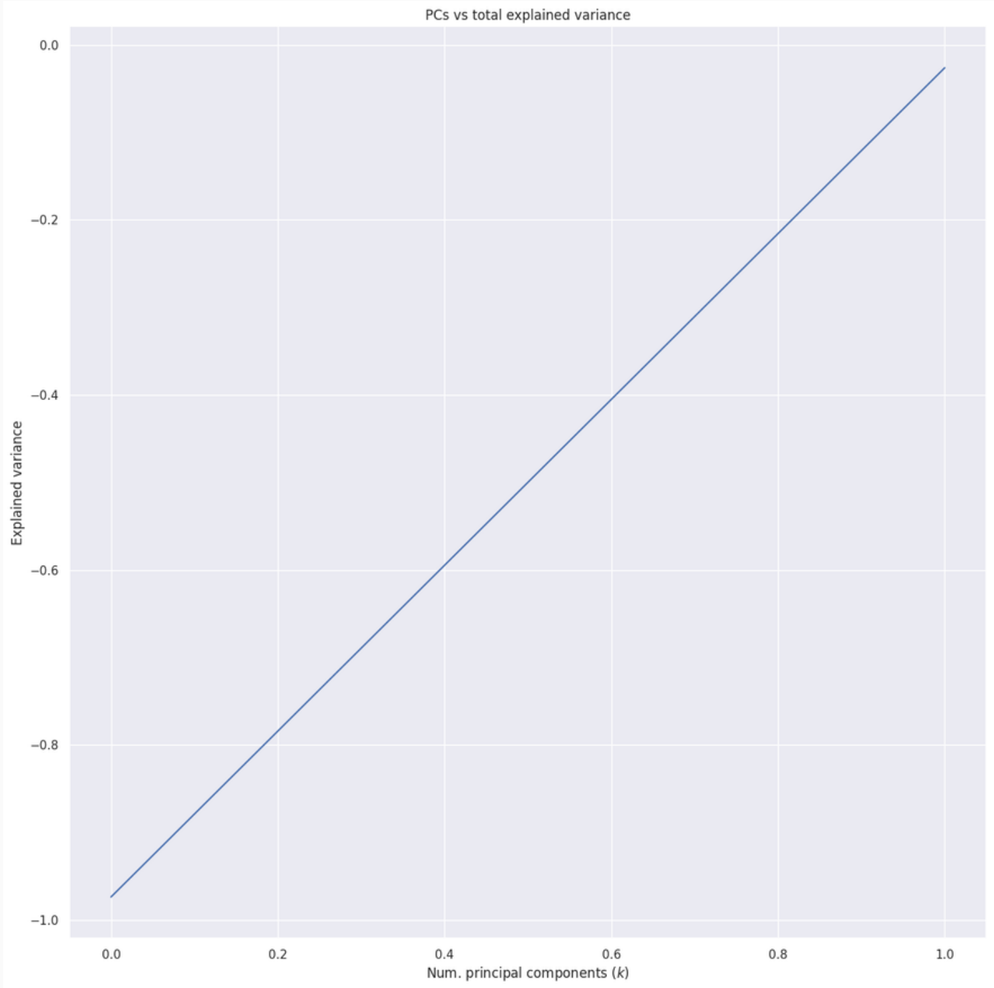
```
                0         1
0       -1.982232  0.262751
1       -1.982232  0.262751
2       -1.982232  0.262751
8       -1.982232  0.262751
11      -1.982232  0.262751
...           ...       ...
114701   2.888625 -0.157656
114702   2.888625 -0.157656
114703   2.888625 -0.157656
114704   2.851801 -0.195208
114705   2.852165 -0.195571

[38698 rows x 2 columns]
```

Aqui se codifica el “Total expleined Variance” para poder conocer los n_components

```
[49]  1 total_explained_variance = sum(explained_variance[:k])
      2
      3 print(explained_variance)
      4
      5 print("total_explained_variance (k=2):", total_explained_variance)
```

```
[0.9737275 0.0262725]
total_explained_variance (k=2): 1.0
```

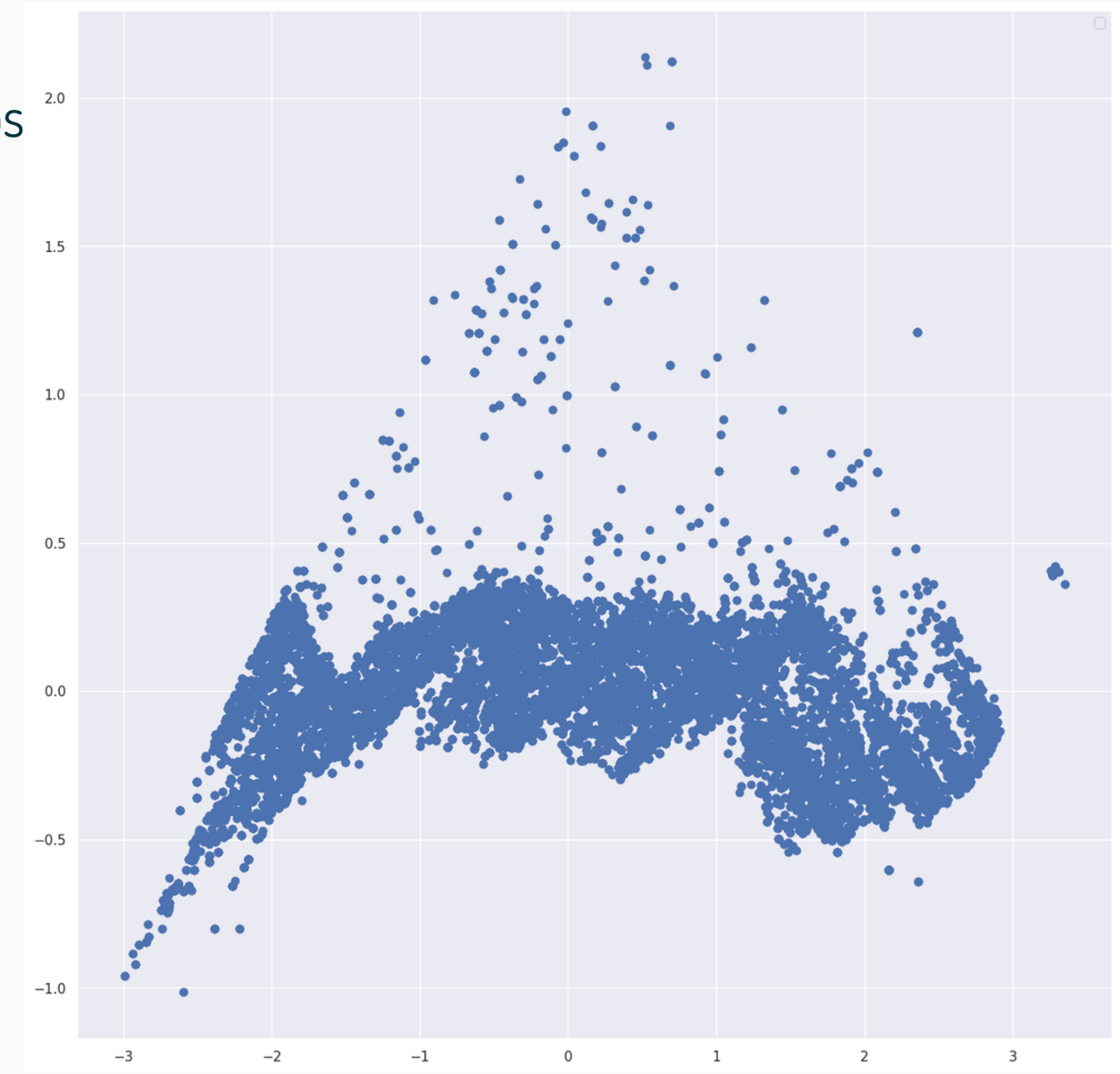
```
[50]  1 plt.plot(-1*explained_variance)
      2 plt.title("PCs vs total explained variance")
      3 plt.xlabel("Num. principal components ($k$)")
      4 plt.ylabel("Explained variance")
      5 plt.show()
```



Y con los n_components identificados lo aplicamos junto a la libreria de PCA para obtener la grafica

```
1 from sklearn.decomposition import PCA
2
3 mypca = PCA(n_components=2)
4 X_pca = mypca.fit_transform(standardized_data)
```

```
1 plt.scatter(X_pca[:,0], X_pca[:,1], cmap="rainbow")
2 plt.legend()
3 plt.show()
```



Red Neuronal

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras

X = datos[['producto']].values
y = datos['Promedio'].values
caler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

predicciones = model.predict(X_test)

mse = mean_squared_error(y_test, predicciones)
print(f'Mean Squared Error: {mse}')

for i in range(10):
    print(f'Valor real: {y_test[i]}, Predicción: {predicciones[i][0]}')

plt.scatter(y_test, predicciones)
plt.xlabel('Valor Real')
plt.ylabel('Predicción')
plt.title('Predicciones vs Valores Reales')
plt.show()
```